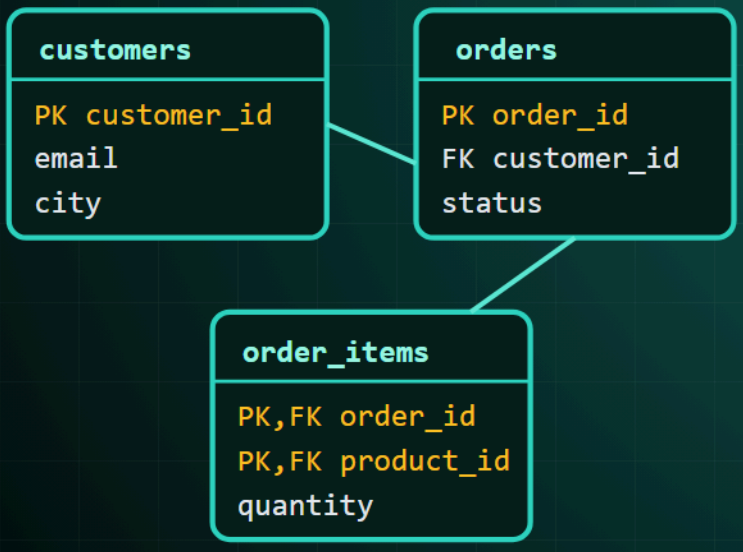



```sql
SELECT c.city,
  SUM(i.quantity * i.unit_price)
FROM  orders o
JOIN  order_items i USING (order_id)
JOIN  customers c USING (customer_id)
WHERE  o.status <> 'cancelled'
GROUP BY  c.city;
```

SQL · DESIGN · STORED PROGRAMS

MySQL for Developers

Learn SQL, design real databases, and program inside MySQL

Joins, indexes, transactions, JSON, procedures, and app code — with exercises and solutions in every chapter

SQL · JOINS · INDEXES · JSON · PROCEDURES

THENEXBOOKS

MySQL for Developers

Learn SQL, Design Real Databases, and Program Inside MySQL
Queries, Joins, Design, Indexes, Transactions, JSON, Stored Programs, and Application Code

First Edition

THENEXBOOKS

MySQL for Developers
Learn SQL, Design Real Databases, and Program Inside MySQL

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. Code examples, the sample database script, and exercise solutions may be used freely for learning and in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Examples target MySQL 8.4 LTS and most also run on MySQL 8.0; always consult the official documentation for the version you use.

MySQL is a trademark of Oracle and/or its affiliates. Other product names are trademarks of their respective owners. Their use in this book is for identification only; this book is independent and is not affiliated with or endorsed by Oracle.

Copperleaf Market, its towns, suppliers, customers, and all sample data are fictional. Any resemblance to real businesses or people is coincidental.

Preface

Almost every application stores its data somewhere, and very often that somewhere is MySQL. Yet many developers learn just enough SQL to get by — a few SELECT statements, an ORM, and hope — and then struggle when reports are wrong, pages get slow, or two customers buy the last item at the same moment. The database is not a black box. Once you understand how to query it, design it, and program it, it becomes one of the most powerful tools you have.

This book is written for developers. We start from zero with tables, rows, and simple queries, and build up step by step: functions and aggregates, joins and subqueries, safe data changes, relational design and normalization, constraints and indexes, transactions and locking. Then we go further with window functions, recursive queries, JSON, full-text search, views, stored procedures, functions, triggers, and events — and finish by using MySQL properly from Python, Node.js, PHP, and Java.

Every example uses the same small, realistic database, so you can run each query yourself and compare your results with the book. Every chapter ends with graded exercises and worked solutions. By the end you will be able to design a schema, write correct and efficient SQL, and build the database layer of a real application with confidence.

Who This Book Is For

- **Beginners** who want to learn SQL and relational databases properly, with MySQL.
- **Web and back-end developers** who use MySQL every day and want to understand it beyond the ORM.
- **Students and career changers** preparing for developer or data roles.
- **Analysts and testers** who need to query application databases accurately.

What You Will Learn

- How to install MySQL and its tools and load a sample database.
- SQL fundamentals: SELECT, filtering, sorting, functions, aggregates, joins, subqueries, and CTEs.
- How to change data safely and design schemas with keys, normalization, and constraints.
- Indexes, transactions, isolation levels, and locking — explained for application developers.
- Window functions, recursive queries, JSON, full-text search, views, and generated columns.
- Stored procedures, functions, triggers, events, security, performance, and application code.

How This Book Is Organized

Part I — Getting Started with MySQL (Chapters 1–5) covers databases, installation, first tables, basic queries, and data types. **Part II — Essential SQL** (Chapters 6–10) covers functions, aggregates, joins, subqueries, and data changes. **Part III — Designing Databases** (Chapters 11–15)

covers relational design, normalization, constraints, indexes, and transactions. **Part IV — Advanced SQL** (Chapters 16–20) covers window functions, recursive CTEs, JSON, full-text search, and views. **Part V — Programming Inside MySQL** (Chapters 21–25) covers stored programs, triggers, events, security, and query performance. **Part VI — MySQL in Applications** (Chapters 26–28) connects MySQL to application code and brings everything together. The appendices provide a cheat sheet, templates and references, a review checklist, and a glossary.

The Running Example: Copperleaf Market

Throughout the book we work on the database of **Copperleaf Market**, a fictional online farm market that delivers groceries from local suppliers. Its categories, products, customers, orders, and reviews are small enough to check by hand but rich enough for real questions — and the complete script to create and fill the database is included, so your results match the book's.

Conventions

SQL keywords appear in uppercase and names in lowercase snake_case. Query results are shown the way the `mysql` command-line client prints them; timings in result footers are illustrative. File captions show where code belongs. Exercises are labeled **[Easy]**, **[Medium]**, or **[Challenge]**.

> **TIP**
> A recommended practice, or a "Try It" prompt to pause and experiment.

> **NOTE**
> Background information or a version difference.

> **WARNING**
> A common mistake that loses data, returns wrong results, or slows things down.

Contents

Preface

PART I — GETTING STARTED WITH MYSQL

1 What MySQL Is and How Relational Databases Work

2 Installing MySQL and Its Tools: Server, MySQL Shell, Workbench, and Docker

3 Your First Database: Creating Schemas, Tables, and Rows

4 Querying Data: SELECT, WHERE, ORDER BY, and LIMIT

5 Data Types in Practice: Numbers, Strings, Dates, and NULL

PART II — ESSENTIAL SQL

6 Functions and Expressions: String, Numeric, Date, and Conditional Functions

7 Aggregating Data: GROUP BY, HAVING, and Summary Reports

8 Joining Tables: Inner, Outer, Self, and Cross Joins

9 Subqueries and Common Table Expressions

10 Changing Data Safely: INSERT, UPDATE, DELETE, and Upserts

PART III — DESIGNING DATABASES

11 Relational Design: Entities, Keys, and Relationships

12 Normalization and When to Denormalize

13 Constraints and Data Integrity: Keys, Foreign Keys, UNIQUE, CHECK, and Defaults

14 Indexes for Developers: How They Work and How to Choose Them

15 Transactions and Concurrency: ACID, Isolation Levels, and Locking

PART IV — ADVANCED SQL

16 Window Functions: Rankings, Running Totals, and Moving Averages

17 Recursive CTEs and Hierarchical Data

18 JSON in MySQL: Storing, Querying, and Indexing Documents

19 Full-Text Search and Working with Text

20 Views and Generated Columns

PART V — PROGRAMMING INSIDE MYSQL

21 Stored Procedures: Parameters, Variables, and Control Flow

22 Stored Functions, Error Handling, and Cursors

23 Triggers and Scheduled Events

24 Users, Privileges, and Security for Developers

25 Query Performance for Developers: EXPLAIN and Common Fixes

PART VI — MYSQL IN APPLICATIONS

26 Connecting from Applications: Python, Node.js, PHP, and Java

27 Imports, Exports, Backups, and Schema Migrations for Developers

28 Case Study: Building the Copperleaf Market Database

APPENDICES

A MySQL SQL Cheat Sheet

B Templates and Reference

C Schema and Query Review Checklist

D Glossary

PART I

Getting Started with MySQL

What relational databases are, installing MySQL and its tools, your first tables, basic queries, and data types.

1 · What MySQL Is and How Relational Databases Work

2 · Installing MySQL and Its Tools

3 · Your First Database

4 · Querying Data

5 · Data Types in Practice

CHAPTER 1 · PART I — GETTING STARTED WITH MYSQL

What MySQL Is and How Relational Databases Work

Before we type a single command, we need a clear picture of what a relational database is, how SQL talks to it, and where MySQL fits. This chapter builds that picture and introduces Copperleaf Market, the small online farm market whose database you will work on for the rest of the book.

Why Applications Need a Database

Almost every application you will build needs to remember things. A to-do app remembers tasks, a shop remembers products and orders, a game remembers players and scores. The simplest way to remember data is to write it to a file: a CSV export, a JSON document, a spreadsheet. That works for a weekend project, and then it stops working.

Picture a small grocery business that keeps its orders in a spreadsheet. Each row holds the customer's name, email address, the products bought, the prices, and the order date. Soon the problems start to pile up:

- **Duplication.** A customer who orders ten times has their email address typed ten times. When it changes, someone has to find and fix all ten copies, and one copy is always missed.
- **Inconsistency.** One row says "Riverside", another says "riverside ", a third says "Riversde". A report that counts customers per town is now wrong.
- **Concurrency.** Two people edit the file at the same time and one person's changes silently overwrite the other's.
- **Questions are hard to ask.** "Which products did gold-tier customers in Lakeview buy last month?" means filtering, copying, and cross-referencing by hand.
- **Safety.** A crash halfway through saving can leave the file half-written. There is no way to say "apply all of these changes or none of them".

A **database management system** (DBMS) is software built to solve exactly these problems. It stores data in a structured, durable way, enforces rules about what valid data looks like, lets many users read and write at the same time without trampling each other, and answers questions through a query language. The data itself, organized for a purpose, is the **database**; the software that manages it is the DBMS. In everyday speech people say "the database" for both, and we will too when the meaning is obvious.

Relational Databases in Plain Terms

There are many kinds of DBMS: document stores, key-value stores, graph databases, time-series databases and more. The most widely used family for business applications is the **relational database management system** (RDBMS). The relational model was described by Edgar F. Codd at IBM in 1970, and its central idea is simple: store data in **tables** (formally called *relations*) and connect tables through shared values instead of pointers or nested structures.

MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server, SQLite and MariaDB are all relational database systems. They differ in features and details, but the core ideas you learn in this book (tables, keys, joins, transactions, SQL) carry over to all of them.

Tables, rows, and columns

A **table** holds data about one kind of thing: categories, products, customers, orders. Each **column** describes one attribute of that thing and has a name and a **data type** (whole number, decimal, text, date, and so on). Each **row** is one instance of the thing: one category, one product, one customer. Here is the `categories` table from the Copperleaf database, exactly as the `mysql` client prints it:

```
mysql> SELECT * FROM categories;
+-------------+--------------+
| category_id | name         |
+-------------+--------------+
|           2 | Bakery       |
|           5 | Beverages    |
|           3 | Dairy & Eggs |
|           4 | Pantry       |
|           1 | Produce      |
+-------------+--------------+
5 rows in set (0.00 sec)
```

The table has two columns, `category_id` and `name`, and five rows. Unlike a spreadsheet, every value in a column must match the column's data type: you cannot put the word "five" into a numeric column, and you cannot leave out a value that the table declares as required. That strictness is a feature. It means the data you read back is the shape you expect, every time.

Two more properties matter. First, the rows of a table have no built-in order. The categories above came back in name order only because MySQL happened to read them through the index on `name`. If you want results sorted, you ask for it (Chapter 4 shows how with `ORDER BY`). Second, each value in a cell should be *atomic*: one email address per cell, not a comma-separated list of three. We return to that rule when we study design in Chapters 11 and 12.

Keys: how rows are identified and connected

A **primary key** is a column (or a set of columns) whose value uniquely identifies each row. In `categories` the primary key is `category_id`: there is exactly one category with id 3, and it is "Dairy & Eggs". A primary key can never be empty (NULL) and can never repeat. Copperleaf uses *surrogate*

keys, plain numbers generated by the database, rather than *natural keys* such as a product name, because names change and numbers do not have to.

A **foreign key** is a column in one table that refers to the primary key of another table. Every product belongs to a category, so `products` has a `category_id` column whose values must exist in `categories.category_id`. That shared value is the whole "relationship" in a relational database. Here are the items of order 1001:

```
mysql> SELECT order_id, product_id, quantity, unit_price
    -> FROM order_items
    -> WHERE order_id = 1001;
+----------+------------+----------+------------+
| order_id | product_id | quantity | unit_price |
+----------+------------+----------+------------+
|     1001 |          1 |        2 |       4.50 |
|     1001 |          4 |        1 |       6.50 |
|     1001 |          6 |        1 |       5.40 |
+----------+------------+----------+------------+
3 rows in set (0.00 sec)
```

The row with `product_id` 4 says "this order contains one unit of product 4". To learn that product 4 is the Sourdough Loaf, the database follows the foreign key to the `products` table. Combining tables along keys like this is called a **join**, and Chapter 8 is devoted to it.

Notice that `order_items` has no single id column. Its primary key is the *pair* (`order_id`, `product_id`), a **composite key**: one order can contain many products, one product can appear in many orders, but a given product appears at most once per order. Finally, a **unique key** guarantees that a non-primary column has no duplicates, like the customer email address. Keys are the database's way of stating facts about your business that must always stay true.

> **NOTE**
>
> The DBMS enforces keys for you. If an application tries to insert an order item for product 99 and no product 99 exists, MySQL rejects the row. A spreadsheet would accept it happily and you would discover the mistake weeks later in a broken report.

SQL and Its Sub-Languages

You talk to a relational database in **SQL**, the Structured Query Language. People pronounce it "S-Q-L" or "sequel"; both are fine. SQL is *declarative*: you describe the result you want, not the steps to compute it. When you write

```
SELECT name, unit_price
FROM products
WHERE unit_price > 6.00;
```

you are not telling MySQL to open a file, loop over records and compare numbers. You are describing a result: "the name and price of every product that costs more than six dollars". A component called the **query optimizer** decides how to produce it, for example whether to scan the whole table or use an index. That separation is what makes SQL so productive: the same short statement works on twelve rows or twelve million.

SQL is standardized by ISO, and every database implements the standard with its own extensions and gaps. This book teaches the MySQL dialect and points out MySQL-specific features when they appear. SQL statements are usually grouped into four sub-languages:

Sub-language	Purpose	Typical statements	Where in this book
DDL — Data Definition Language	Create and change the structure of databases, tables, indexes, views	`CREATE`, `ALTER`, `DROP`, `TRUNCATE`	Chapters 3, 13, 14, 20
DML — Data Manipulation Language	Read and change the rows inside tables	`SELECT`, `INSERT`, `UPDATE`, `DELETE`	Chapters 4–10
DCL — Data Control Language	Control who may do what	`GRANT`, `REVOKE`	Chapters 2 and 24
TCL — Transaction Control Language	Group changes so they succeed or fail together	`START TRANSACTION`, `COMMIT`, `ROLLBACK`, `SAVEPOINT`	Chapter 15

Some texts put `SELECT` in its own group, DQL (Data Query Language), because it only reads. The grouping is a teaching aid rather than a strict rule, but the categories help you reason about risk. A `SELECT` never changes data. A DML statement such as `UPDATE` changes rows. A DDL statement such as `DROP TABLE` changes structure, and in MySQL most DDL statements commit immediately and cannot be rolled back. One example of each, all of which you will write yourself in later chapters (just read them for now; do not run them against `copperleaf`):

```
-- DDL: define a structure
CREATE TABLE categories (
  category_id  INT UNSIGNED NOT NULL AUTO_INCREMENT,
  name         VARCHAR(50)  NOT NULL,
  PRIMARY KEY (category_id)
);

-- DML: add and read rows
INSERT INTO categories (name) VALUES ('Produce');
SELECT category_id, name FROM categories;

-- DCL: give a user permissions
GRANT SELECT ON copperleaf.* TO 'copperleaf_report'@'localhost';

-- TCL: make several changes atomic
```

```
START TRANSACTION;
UPDATE products SET stock_qty = stock_qty - 2 WHERE product_id = 1;
COMMIT;
```

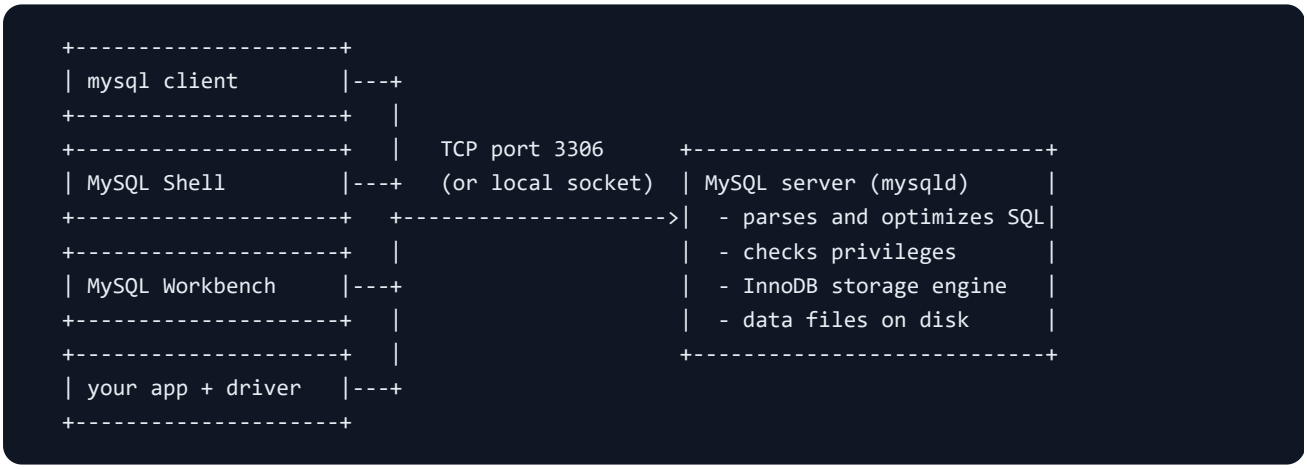
> **TRY IT**
>
> Look at the four statement groups above and classify these without peeking at the table: `ALTER TABLE`, `DELETE`, `REVOKE`, `ROLLBACK`. Then ask yourself which of them could lose data if run by mistake. (Answers: DDL, DML, DCL, TCL; `ALTER TABLE` and `DELETE` are the dangerous ones.)

The Client–Server Model

MySQL is a **client–server** database. The **server** is a long-running program named `mysqld` (the "d" stands for daemon). It owns the data files on disk, manages memory and caches, checks permissions, and executes SQL. It listens for connections, by default on TCP port **3306**, and on Unix-like systems also on a local socket file.

A **client** is any program that connects to the server, sends SQL, and receives results. You will use several clients in this book:

- the `mysql` command-line client, a classic terminal tool that ships with the server;
- **MySQL Shell** (`mysqlsh`), a modern shell with SQL, JavaScript and Python modes;
- **MySQL Workbench**, a graphical tool with a SQL editor and schema designer;
- your own applications, through a driver or connector for Python, Node.js, PHP, Java and others (Chapter 26).



Every client follows the same conversation. It opens a connection, authenticates with a user name and password, optionally chooses a default database, sends statements, reads result sets or status messages, and finally disconnects. Because the server is separate, many clients can work at once: the web shop, a reporting job, and you in a terminal can all use the same `copperleaf` database simultaneously, and the server keeps their work consistent.

The server and the clients can run on the same machine (typical on your laptop) or on different machines across a network (typical in production, where the application servers connect to a dedicated database host or a managed cloud service). The SQL is the same either way; only the connection details change: host name, port, user, and password.

Inside the server, MySQL uses a pluggable **storage engine** layer. The default engine, and the only one this book uses, is **InnoDB**. It provides transactions, row-level locking, crash recovery, and foreign keys. You will occasionally see older tutorials using MyISAM; it does not support transactions or foreign keys, and there is rarely a reason to choose it for new application tables.

What MySQL Is

MySQL is an open-source relational database management system. It was first released in the mid-1990s by the Swedish company MySQL AB, which Sun Microsystems acquired in 2008; Oracle acquired Sun in 2010 and has developed MySQL since then. The name combines "My", the name of co-founder Michael Widenius's daughter, with "SQL".

MySQL earned its popularity by being free, fast for typical web workloads, easy to install, and available everywhere: on every major operating system, in every major programming language through drivers, and as a managed service from the large cloud providers.

Editions

- **MySQL Community Edition** is the free, open-source edition, licensed under the GNU General Public License (GPL). Everything in this book works with it, and it is what you install in Chapter 2.
- **MySQL Enterprise Edition** is Oracle's commercial subscription. It adds extra tools and plugins (for example for backup, auditing, monitoring, and encryption-related features) plus Oracle support. The SQL language is the same.
- **MySQL HeatWave** is Oracle's managed cloud database service built on MySQL, with additional analytics and machine-learning capabilities. Other cloud providers offer their own managed MySQL services as well.

Licensing details matter if you plan to embed and redistribute MySQL inside a commercial product. For the usual case, an application that connects to a MySQL server, they rarely come up, but check the current terms with your legal team if in doubt.

Release tracks: LTS and Innovation

For many years MySQL had one main line per major version, and the 8.0 series received both bug fixes and new features through its patch releases. Starting in 2023, Oracle changed the model and now publishes two kinds of releases:

- **Long-Term Support (LTS)** releases are meant for production systems that value stability. Once an LTS series is out, its updates focus on bug and security fixes rather than new features, and it is supported for several years. **MySQL 8.4 is the first LTS release**, and it is the version this book targets.
- **Innovation** releases (the 8.1 to 8.3 releases and the 9.x series) arrive more frequently, carry new features sooner, and each is supported only until the next one replaces it. They are aimed at people who want the newest features and can upgrade often.

The features of the Innovation releases are eventually collected into the next LTS series. Exact support dates and the numbering of future LTS releases are announced by Oracle, so check the official MySQL documentation and lifecycle pages for your version rather than relying on dates in any book.

For you as a developer the practical advice is simple: use the LTS release unless you need a specific newer feature, and keep your development machine on the same major version as production. Almost every example in this book also works on MySQL 8.0, which is still common in existing systems; where a feature needs a minimum version, we say so.

> **WARNING**
>
> MySQL and MariaDB are related but not interchangeable. MariaDB began as a fork of MySQL and still shares a lot of syntax, but the two have diverged over the years: JSON support, some functions, authentication, and system tables differ. This book is about MySQL. If your production system runs MariaDB, test every example there before relying on it.

Where MySQL Is Used

MySQL is a general-purpose transactional database: it shines when many users read and write small amounts of data at a time. Typical homes for it include:

- **Web applications.** MySQL is the "M" in the classic LAMP stack (Linux, Apache, MySQL, PHP/Perl/Python), and popular content management systems such as WordPress, Drupal, and Joomla use MySQL or a compatible database by default.
- **Online stores and SaaS back ends.** Catalogs, carts, orders, accounts, subscriptions, and billing records are classic relational data.
- **Managed cloud databases.** Amazon RDS and Amazon Aurora (MySQL-compatible edition), Azure Database for MySQL, Google Cloud SQL, and Oracle's MySQL HeatWave all let teams run MySQL without managing the server themselves.
- **Internal tools.** Admin panels, inventory systems, ticketing, and reporting databases.

MySQL is less suited to some jobs: heavy analytics over billions of rows is usually done in a column-oriented warehouse, and huge binary files belong in object storage with only their metadata in the database. Even then, the transactional system that feeds those tools is very often a relational database like MySQL.

Meet Copperleaf Market

Throughout this book you work as a new developer at **Copperleaf Market**, a fictional online farm market. Copperleaf delivers groceries from local suppliers to customers in three towns: **Riverside**, **Lakeview**, and **Hillcrest**. The team is small:

- **Maya Lindqvist**, the founder, who wants reports on sales and customers;
- **Omar Haddad**, the operations manager, who cares about stock, suppliers, and deliveries;
- **Tess Moreno**, the lead developer, who reviews your SQL and owns the schema.

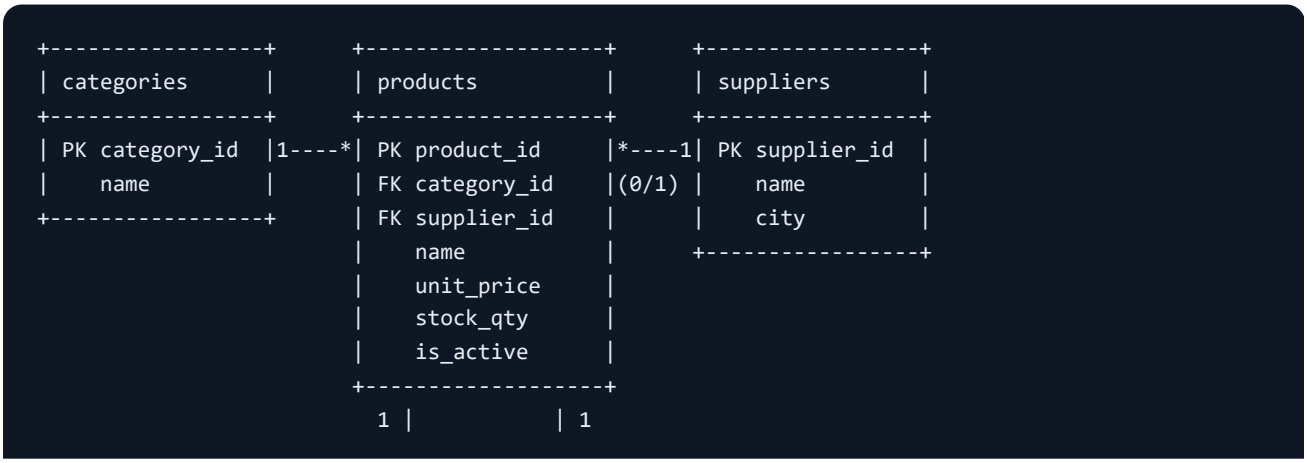
Their database is called `copperleaf`. You load it from the script `copperleaf.sql` in Chapter 2, and every example, exercise, and result in this book is computed from that data. Because the data set is small, you can always check a result by eye, which is the best way to build confidence in a new query.

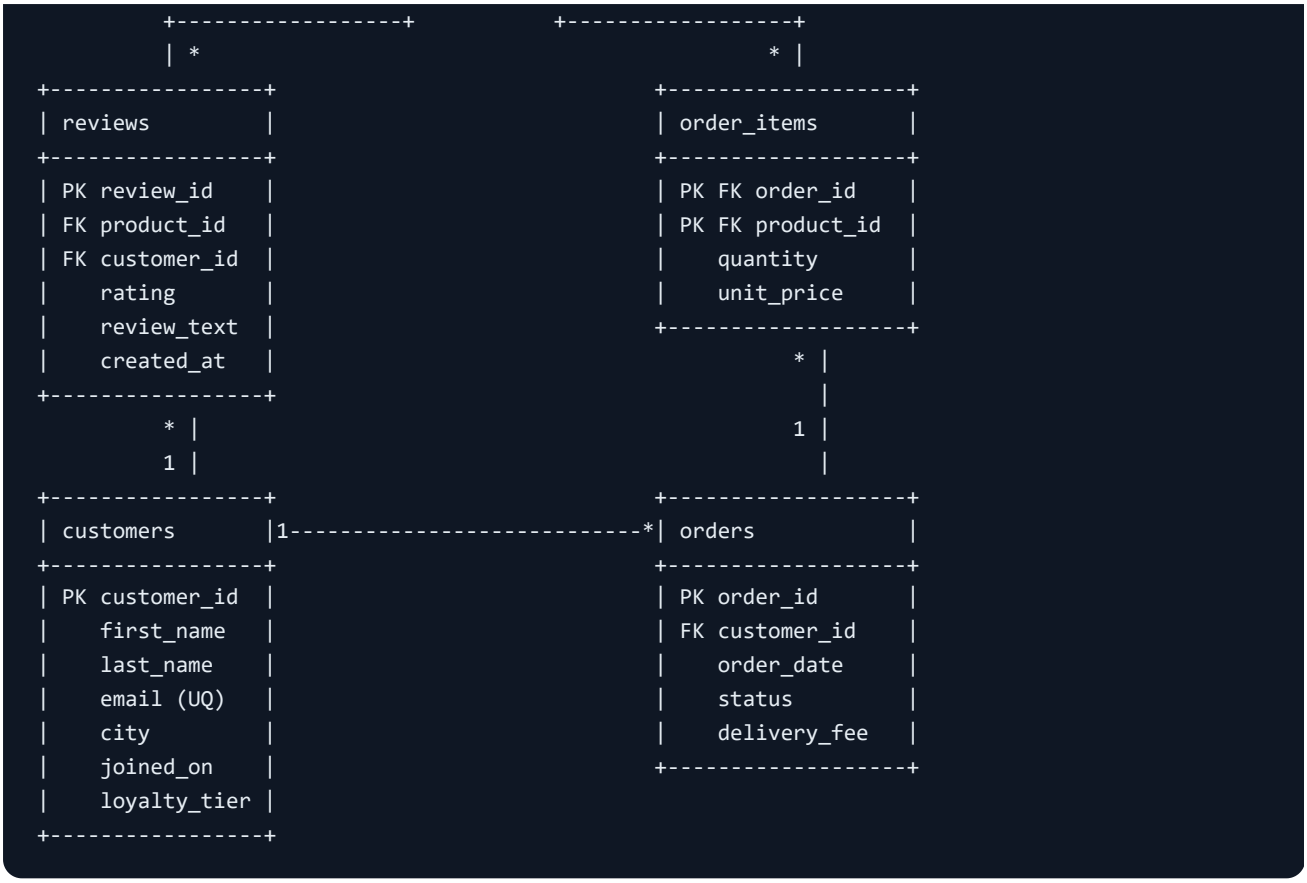
The seven tables

Table	What one row means	Rows	Primary key
`categories`	A product category such as Produce or Bakery	5	`category_id`
`suppliers`	A local farm or producer	4	`supplier_id`
`products`	Something Copperleaf sells, with price and stock	12	`product_id`
`customers`	A registered customer and loyalty tier	8	`customer_id`
`orders`	One checkout: date, status, delivery fee	12	`order_id`
`order_items`	One product line within an order	26	`order_id` + `product_id`
`reviews`	A customer's 1–5 star review of a product	8	`review_id`

The ER diagram

An **entity–relationship (ER) diagram** shows tables as boxes and relationships as lines. In the diagram below, `PK` marks primary key columns and `FK` marks foreign keys. A `1` at one end of a line and a `*` at the other mean "one to many": one category has many products, one customer has many orders.





Read the diagram out loud and it describes the business. A category has many products. A supplier provides many products, but a product may have no supplier: `supplier_id` is optional, marked `(0/1)`, and Rolled Oats (product 10) really has none. A customer places many orders. An order contains many order items, and each order item points to exactly one product. Customers write reviews of products, so `reviews` has two foreign keys.

Business rules worth remembering

- Delivery is free when an order's item subtotal is $25.00 or more; otherwise the delivery fee is $3.00. The fee charged is stored per order in `orders.delivery_fee`.
- `order_items.unit_price` is the price *at the time of the order*. Order 1002 bought House Blend Coffee at 11.90; the current price in `products` is 12.50. Storing the historical price is deliberate, not a mistake.
- An order moves through the statuses `pending`, `paid`, `shipped`, and `delivered`, or ends as `cancelled`. Revenue reports exclude cancelled orders unless we say otherwise.
- Customers have a loyalty tier of `standard`, `silver`, or `gold`; in the base data the tiers carry no discount.

The data also contains a few deliberate oddities that make good lessons later: Rolled Oats has no supplier, Cold-Pressed Apple Juice (product 12) is inactive, out of stock, and has never been ordered, and customer Hana Sato has no city on file and has not placed an order yet. When you learn about NULL values and outer joins, these rows are where the interesting results come from.

> **TRY IT**
>
> Using only the ER diagram, trace how you would answer "Which supplier's products did Amara Osei order?" Write down the chain of tables and the key columns you would follow. (One answer: `customers` → `orders` by `customer_id` → `order_items` by `order_id` → `products` by `product_id` → `suppliers` by `supplier_id`.)

How to Use This Book

The book is organized in six parts. **Part I** (Chapters 1–5) gets MySQL running and teaches you to create tables and query them. **Part II** (Chapters 6–10) covers the everyday SQL every developer needs: functions, aggregation, joins, subqueries, and safe data changes. **Part III** (Chapters 11–15) is about design: keys, normalization, constraints, indexes, and transactions. **Part IV** (Chapters 16–20) covers advanced SQL such as window functions, recursive queries, JSON, full-text search, and views. **Part V** (Chapters 21–25) is programming inside MySQL: stored procedures and functions, triggers, events, privileges, and query performance. **Part VI** (Chapters 26–28) connects MySQL to real applications and closes with a case study that rebuilds the Copperleaf database from scratch. The appendices hold a cheat sheet, templates, a review checklist, and a glossary.

If you are new to databases, read Parts I and II in order; each chapter builds on the previous one. If you already write basic SQL, skim Part I for the setup and conventions, then jump to whatever you need.

Conventions

- SQL keywords are written in UPPERCASE and identifiers in lowercase `snake_case`: `SELECT name FROM products`. MySQL itself does not care about keyword case.
- Statements typed at the `mysql>` prompt are shown with the prompt; continuation lines start with `->`, just as the client prints them. Do not type the prompts.
- Results appear in the `mysql` client's boxed table format, followed by a footer such as `5 rows in set (0.00 sec)`. The time on your machine will differ; timings in this book are illustrative.
- Passwords in examples are placeholders such as `change-me`. Never use them for real.
- Callout boxes mark **Tips**, **Notes**, **Warnings**, and **Try It** prompts that ask you to run or adapt something yourself.

Learning actively

SQL is a skill you learn with your fingers. Every chapter ends with practice exercises labeled Easy, Medium, or Challenge, followed by worked solutions to a selection of them. Try each exercise before reading the solution, and when your answer differs, run both and compare the results. A query that returns the right numbers for the wrong reason will eventually return the wrong numbers.

Finally, you can always rebuild the database: running `copperleaf.sql` again drops the `copperleaf` database and recreates it with the original data. So experiment freely. Break things, read the error messages, and reload. That is exactly how professional developers learn a new database.

Practice Exercises

You install MySQL in the next chapter, so some of these exercises are about reading and reasoning. Come back and run the SQL ones after Chapter 2.

1. **[Easy]** In your own words, explain the difference between a database and a database management system.
2. **[Easy]** Name the primary key of each of the seven Copperleaf tables. Which one is a composite key?
3. **[Easy]** Classify each statement as DDL, DML, DCL, or TCL: `CREATE TABLE`, `INSERT`, `GRANT`, `COMMIT`, `DROP DATABASE`, `UPDATE`.
4. **[Easy]** After installing MySQL, list every supplier with its city.
5. **[Easy]** What default TCP port does the MySQL server listen on, and what is the server program called?
6. **[Medium]** List all foreign keys in the Copperleaf schema, writing each one as `table.column → table.column`.
7. **[Medium]** Using the diagram, explain why a `product_id` column in `order_items` is better than storing the product name there.
8. **[Medium]** Show the items of order 1002 and explain why the coffee price differs from today's catalog price.
9. **[Medium]** A colleague suggests installing an Innovation release on the production server "because it is newer". Write two sentences explaining the trade-off.
10. **[Challenge]** Order 1003 was charged no delivery fee. Using the order items, check that this follows the business rule.
11. **[Challenge]** Sketch, in ASCII, how you would extend the ER diagram to record which delivery driver handled each order. Which table gets the new foreign key, and which side of the relationship is "many"?

Selected Solutions

Exercise 4

```
SELECT supplier_id, name, city
FROM suppliers;
+-------------+--------------------+-----------+
| supplier_id | name               | city      |
+-------------+--------------------+-----------+
```

```
|           1 | Green Hollow Farm  | Riverside |
|           2 | Stonebridge Bakery | Lakeview  |
|           3 | Meadowlark Dairy   | Hillcrest |
|           4 | Ridgeback Roasters | Lakeview  |
+-------------+--------------------+-----------+
4 rows in set (0.00 sec)
```

Two suppliers are in Lakeview and none of the four lacks a city: the `city` column of `suppliers` is declared `NOT NULL`.

Exercise 6

The schema has seven foreign keys: `products.category_id → categories.category_id`, `products.supplier_id → suppliers.supplier_id`, `orders.customer_id → customers.customer_id`, `order_items.order_id → orders.order_id`, `order_items.product_id → products.product_id`, `reviews.product_id → products.product_id`, and `reviews.customer_id → customers.customer_id`. Of these, only `products.supplier_id` allows NULL.

Exercise 8 (Medium)

```
SELECT order_id, product_id, quantity, unit_price
FROM order_items
WHERE order_id = 1002;
+----------+------------+----------+------------+
| order_id | product_id | quantity | unit_price |
+----------+------------+----------+------------+
|     1002 |          7 |        2 |       4.20 |
|     1002 |         11 |        1 |      11.90 |
+----------+------------+----------+------------+
2 rows in set (0.00 sec)
```

Product 11 is House Blend Coffee. The order stored 11.90 because that was the price on the day of the order; the product's current price of 12.50 lives in `products.unit_price`. Copying the price into the order line keeps historical orders correct when prices change.

Exercise 10 (Challenge)

```
SELECT order_id, product_id, quantity, unit_price
FROM order_items
WHERE order_id = 1003;
+----------+------------+----------+------------+
| order_id | product_id | quantity | unit_price |
+----------+------------+----------+------------+
|     1003 |          3 |        1 |       2.90 |
|     1003 |          8 |        1 |       6.90 |
|     1003 |          9 |        2 |       8.75 |
+----------+------------+----------+------------+
3 rows in set (0.00 sec)
```